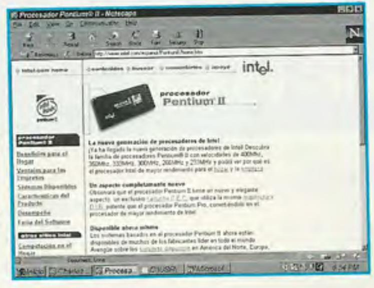


El código del Katmai 3D al descubierto

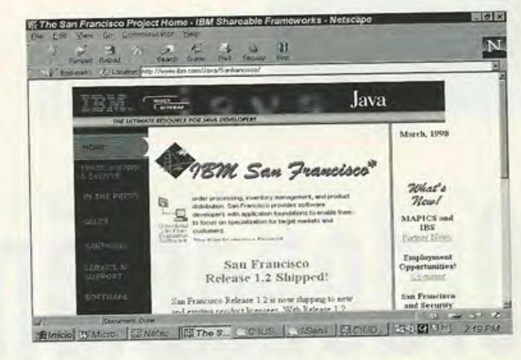


Según fuentes independientes ajenas a la gigantesca compañía Intel, que por cierto, últimamente ha tenido acusaciones de monopolio respecto a su política de productos, se ha podido romper el código por un programador independiente de su nuevo y esperado procesador Katmai.
Katmai es un avanzado procesador de 32 bits que incorpora toda la tecnología necesaria para soportar las futuras aplicaciones y juegos 3D. Se denomina también como MMX2 porque se considera como la continuación de este chip que tan corta vida ha tenido. Incorpora 70 nuevas instrucciones que complementan a las 57 instrucciones de extensión multimedia que incorporaba el MMX original.
Pues bien, toda esta tecnología ha sido *descubierta* por alguien ajeno a Intel, con el consiguiente perjuicio que puede ocasionar para los futuros planes comerciales de esta compañía.

Windows NT y el año 2000

Debido a una curiosa historia comercial cuyo desarrollo sería imposible de explicar en tan poco espacio, Microsoft ha podido comprobar que su Windows NT 4.0 no es muy seguro que digamos con la llegada del año 2.000 así que ha tenido que chequear los numerosos parches que solucionan el problema más temido de fin de siglo y que se pueden encontrar en una página web especial. De todas formas, la compañía ha asegurado repetidamente que el problema del año 2.000 no tendrá nada que hacer con sus nuevos sistemas NT 5.0 y Windows 98.

IBM en San Francisco

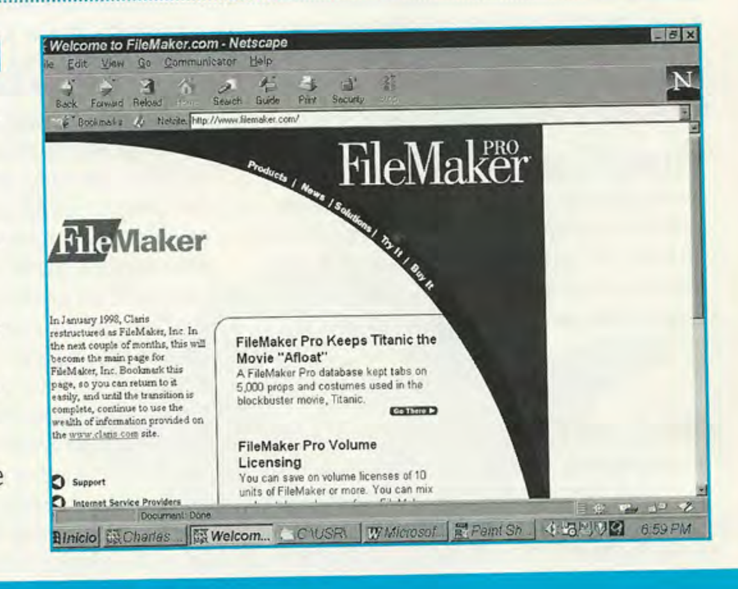


No, no es que IBM se cambie de sede, sino que su nueva aplicación tiene el mismo nombre que la bonita ciudad americana. Sus promotores consideran que será toda una revolución e incorporará la última tecnología en Java.

Sumario

FileMaker Pro Developer Edition es 100% puro Java

FileMaker Inc. ha anunciado que su programa FileMaker Pro. 4.0 Developer Edition ha obtenido el certificado de la división de Java Software de Sun Microsystems que le acredita como 100% puro Java. FileMaker Pro 4.0, cuya comercialización está prevista en julio de 1998, proporcionará a los desarrolladores una solución completamente independiente de la plataforma para desarrollar soluciones de bases de datos.
FileMaker Developer Edition presenta varios API1S que permiten al desarrollador escribir funciones externas para las bases de datos FileMaker Pro 4.0. Esta capacidad amplía en gran medida, la base funcional de FileMaker Pro 4.0. Además, el Developer Edition incorpora una librería de Java Classes que permite el acceso directo a las bases de datos FileMaker Pro a través de todos los sistemas operativos que soportan la plataforma Java entre los que se incluyen los entornos Unix y Solaris. FileMaker Inc. se une a las más de 100 aplicaciones que también han obtenido la certificación 100% puro Java.





En nuestro CD de portada incluimos el siguiente material:

- Las fuentes de código de los ejemplos comentados en 3D manía.
- COOL EDIT: Editor de samples para generar efectos de sonido de calidad.
- Ejemplos del curso de DIV Games Studio.

Autor: Jesús de Santos

Ordenación de Polígonos

Tras los temas tratados en los últimos meses es el momento de visualizar figuras sólidas en tres dimensiones. En el capítulo de este mes vamos a analizar algunos de los métodos que existen para nuestro propósito.

Hasta ahora hemos estado trabajando con figuras en alámbrico (sólo pintábamos las aristas de los triángulos). De este modo, hemos evitado los problemas que conlleva una representación de una figura sólida. Resueltos los principales pormenores que se han encontrado a lo largo de los últimos meses (se aprendió a usar matrices y cámaras, a ocultar caras y a recortar polígonos en 3d) ha llegado la hora de mejorar el sistema de visualización. En los primeros capítulos de esta sección se explicaron a fondo algunas técnicas para pintar triángulos con textura y, en general, para interpolar cualquier propiedad a lo largo de la superficie de un polígono de n-lados.
De momento, siguiendo con la filosofía de esta sección, que es la de avanzar poco a poco y explicar todos los pasos que demos sin meter nada que no se haya analizado antes, no vamos a pintar polígonos texturados. Nos podemos dar por satisfechos si logramos pintar un objeto en modo 'Flat' (color constante por cada triángulo). El objeto lo leeremos, por supuesto, con el lector de ficheros 3d que desarrollamos el mes pasado. Según vayamos necesitando nuevas cosas de nuestro formato g3d iremos modificando el conversor. Este mes, no vamos a realizar ninguna modificación sobre él.

Y LLEGO EL COLOR

No detallaremos aquí a fondo toda la rutina de pintado de triángulo debido a la sencillez de ésta. Las bases del pintado de triángulos se explicaron ampliamente en los números 1 y 2 de la revista. La rutina que vamos a emplear es la siguiente:

```
void triangle_flat(......);
```

Además, definimos una estructura de tipo vértice donde pasamos información sobre cada uno de los vértices. Recordemos que estamos trabajando con coordenadas ya proyectadas en pantalla y, por tanto, en 2D. La componente *z* del vértice antes de ser proyectado también la guardamos pues será necesaria, como ya veremos:

```
BEGIN VRT
    x       :       4 bytes (float)
    y       :       4 bytes (float)
    z       :       4 bytes (float)
END VRT
```

Según vayamos necesitando más información iremos ampliando esta estructura. Así, por ejemplo, en próximos artículos precisaremos información adicional que haga referencia a las coordenadas de textura del vértice, la iluminación, ...
De la rutina no vamos a comentar nada más. Simplemente destacar el hecho de que los vértices que le pasemos deben estar ordenados en el sentido de las agujas del reloj (lo cual no es ningún problema, pues es el convenio que estamos siguiendo para ocultar caras en nuestro motor) y que se puede pasar un número de vértices superior a tres. Es decir, no se limita únicamente a pintar triángulos, sino que podemos pintar cualquier polígono de n lados que sea convexo.
El código del procedimiento está completo en el ejemplo que este mes acompaña al artículo.

OCULTACION DE CARAS

Al pintar caras sólidas nos vamos a encontrar con un problema importante que hasta ahora no había aparecido. Si tomamos el visor del mes pasado y pintamos las caras con la rutina de polígonos que hemos descrito más arriba, observaremos que las caras del objeto se solapan de un modo incorrecto. Esto es lógico debido a que unas caras están más cerca que otras de la cámara y, por tanto, pueden tapar a caras que estén más lejos de la cámara. Vamos a tener que resolver este problema teniendo en cuenta la profundidad (componente z) de cada uno de los triángulos que pintemos.
Vamos a citar dos técnicas que son muy usadas en estos casos. Existen muchas más pero, debido a que no podemos hablar de todas, tomaremos dos que son bastante representativas. Cada una de ellas tiene sus pros y sus contras. El conocimiento de ambas nos ayudará a saber cuál debemos escoger en cada caso.

- Técnica 1: *ZBUFFER*

Posiblemente, ésta sea una de las técnicas más sencillas. Consiste en tener un bloque de memoria (buffer) donde guardamos la distancia de cada uno de los píxeles de pantalla (componente Z), de ahí el nombre de Zbuffer. Vamos a explicarlo un poco más a fondo.
Tenemos el Zbuffer que suele tener la misma capacidad que la memoria de la zona de visualización. De este modo, para cada píxel de la pantalla tenemos una casilla en el Zbuffer donde podemos guardar la información referente a ese píxel. En el Zbuffer guardamos la distancia del píxel, actualmente pintado en pantalla. Cuando vamos a pintar un nuevo píxel en pantalla comprobamos su distancia con la que está guardada en el Zbuffer. Si la comparación es positiva (existen diversos criterios que ahora veremos) entonces pintamos el nuevo píxel y actualizamos el valor del Zbuffer. En caso de que la comparación sea negativa, no pintamos el píxel pues éste queda detrás del que actualmente hay pintado y no se ve. Destacar que nos estamos refiriendo a la componente *z* del vértice una vez transformado a coordenadas de cámara.
Con lo que hemos visto, podemos deducir que necesitamos la distancia de cada uno de los píxeles que vamos pintando en pantalla con respecto a la cámara. Para esto tenemos que interpolar la componente *z* a lo largo de la superficie del triángulo. Ya vimos cómo se hacía esto. El bucle que queda para una rutina que pinta triángulos en modo *flat* sería algo parecido a esto:

```
for(i=0;i<ancho;i++) {

        if( (cz>>16) < *(zbuffer + i)) {

                *(scan+x1+i) = color;
                *(zbuffer+i) = cz >> 16;

        }

        cz += fdudz;
}
```

Cada vez que borremos la pantalla deberemos borrar también el zbuffer. El valor con el que debemos borrar depende de la función que estemos usando para evaluar las distancias. Lo

normal es emplear una función de comparación. De este modo, el valor de borrado del zbuffer deberá ser la distancia máxima de la escena; es decir, un valor que sabemos que nunca será sobrepasado por ningún polígono que vayamos a pintar. Una de las conclusiones a las que llegamos en los primeros capítulos de esta sección era que si un valor que dependía de 1/z se interpolaba en coordenadas de pantalla, éste lo hacía linealmente. Podemos usar, por tanto, el valor 1/z de cada píxel para comparar y almacenar en el zbuffer. Esto nos va a traer algunas ventajas:

- La interpolación en coordenadas de pantalla es lineal.
- El valor más lejano es 1/**[INFINITO]** que es 0. Éste es el valor que podemos usar para borrar el zbuffer. De este modo, podemos asegurar que no habrá ningún polígono que esté más lejos que este valor.
- Como veremos en próximos artículos, el valor 1/z nos va a ser útil también para la corrección de perspectiva (de nuevo, os remitimos a los primeros números de esta sección). Así, *"matamos dos pájaros de un tiro"*.

Como principal desventaja del empleo de 1/z tenemos que los valores no se almacenan de un modo lineal. Esto tiene el problema de que vértices con z's relativamente grandes toman el mismo valor al invertirlos. La función $f(x) = 1/x$ tiende a 0 en el infinito y según nos acercamos a él los valores tienen a estar más cerca. En general esto no tiene que ser algo muy problemático, pues si un triángulo está muy lejos no importa que se monte con otros también muy lejanos, pues, normalmente, serán triángulos con superficies pequeñas (debido a la distancia).

La precisión que se suele emplear con el Zbuffer suele ser de 16 bits. Un valor de 32 bits (lo que nos permitiría guardar valores de tipo float) puede ser excesivo, no solo por el tamaño sino por las penalizaciones de caché, que se duplican.

La técnica de Zbuffer no suele ser muy apropiada para implementar por software, pues se trata de un proceso lento y que consume mucha memoria. A no ser que sea un caso particular o que la velocidad no importe, el Zbuffer no va a ser una buena solución. En juegos como Quake (que podemos considerar una obra maestra de la programación 3d) se usa un Zbuffer pero bastante restringido. Realmente, sólo se compara el Zbuffer con los objetos del escenario, mientras que éste lleva otro tipo de ordenación.

En cambio, gracias a su sencillez de implementación, el Zbuffer suele venir incorporado en hardware en gran parte de las tarjetas 3d que se encuentran disponibles en el mercado. La memoria para el Zbuffer ya viene en la propia tarjeta junto a la memoria de vídeo y la memoria de texturas.

Por último, sólo destacar que podemos acelerar el pintado de una escena en la que se emplee Zbuffer si ordenamos las caras de adelante a atrás. No olvidemos que cuando estamos rechazando un píxel estamos evitando todo el cálculo necesario de iluminación y textura. Pintando de adelante hacia atrás, las posibilidades de que un píxel no entre en el Zbuffer son mucho mayores. En la técnica 2 vamos a aprender a ordenar polígonos en profundidad.

- Técnica 2: *ALGORITMO DEL PINTOR*

El siguiente algoritmo que vamos a ver recibe su nombre de la técnica que se suele usar en pintura. Así, lo primero que se pinta es el fondo, luego vamos pintando los objetos encima del fondo hasta llegar a los detalles más cercamos. La idea es simple: pintar los objetos desde atrás hacia adelante. Si seguimos esta norma, nuestra escena quedará perfectamente pintada. El primer problema que vamos a encontrar con esta técnica viene con caras que se cortan. Si dos caras se cortan no podemos pintar una antes que la otra pues escojamos la que escojamos siempre una tapa a la otra. En estos casos, lo que se suele hacer es partir los polígonos hasta que se consiga que no se intersecciones.

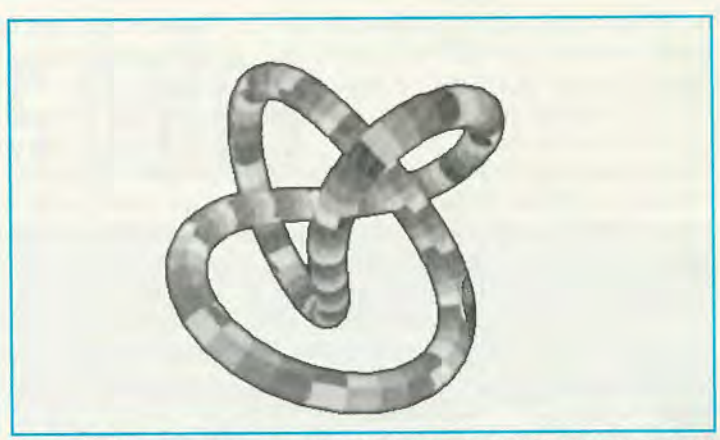


FIGURA 1. ESTOS SON LOS RESULTADOS QUE PODEMOS OBTENER CON LAS TECNICAS DESCRITAS EN EL ARTICULO DE ESTE MES. AUNQUE TODAVIA NO ESTEMOS APLICANDO LUZ, LA SENSACION DE VOLUMEN CONSEGUIDA ES MUCHO MAYOR QUE EN VISORES ANTERIORES.

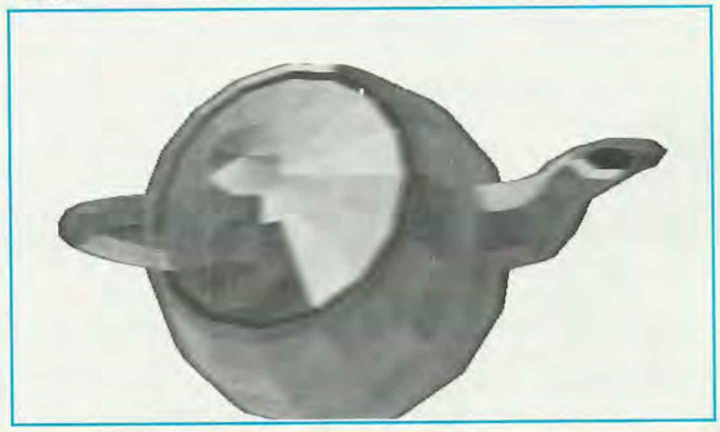
También se pueden llevar a cabo complicadas comprobaciones para pintar las caras correctamente. Con la técnica de Zbuffer este problema es trivial. Nosotros vamos a simplificar mucho más las cosas: en nuestros objetos no vamos a permitir que las caras se corten unas con otras, o si lo hacen, se pintarán mal. Con esta restricción el algoritmo queda mucho más sencillo. Lo podemos resumir del siguiente modo:

1. Almacenar todos los triángulos a pintar en un buffer.
2. Ordenar los triángulos del buffer en profundidad.
3. Pintar los triángulos de atrás a adelante.

El único punto conflictivo es el 2. Los demás son fáciles de llevar a cabo con todo lo que hemos explicado hasta ahora. Lo primero que podemos observar es que esta técnica trabaja con distancias de polígonos (el Zbuffer visto más arriba, trabaja con distancias de píxeles). La distancia de un polígono se puede considerar de varias maneras; por supuesto, todas ellas son aproximaciones. Una de ellas es tomar la distancia del vértice más cercano o más lejano (según convenga en cada caso). En este ejemplo vamos a trabajar con la *z* media del triángulo. Ésta la podemos calcular a partir de los vértices del polígono, sumando las distancias y dividiendo por el número de vértices que forman el polígono. El valor que obtenemos es el valor que utilizaremos como clave de ordenación.

Para ordenar el buffer podemos usar cualquiera de los algoritmos de ordenación conocidos. Así, se puede utilizar el QuickSort que trabaja en tiempo medio O (*n log n*). Un algoritmo ampliamente utilizado en el mundillo de los videojuegos y sobre todo en el de las demos es el que se conoce con el nombre de Radix-Sort. El

FIGURA 2.



RadixSort es un algoritmo de ordenación que trabaja en tiempo 'pseudolineal'. Es decir, en vez de trabajar en O(n) (lineal), trabaja en O(rango) donde el rango es el número de bits de las claves a ordenar.
El mejor modo de ver su funcionamiento es mediante un ejemplo. Radix siempre trabaja con un rango máximo de representación. Tomemos un ejemplo que trabaje con rango 3 bits (ridículo, pero válido para este ejemplo). Como tenemos 3 bits, usamos 2^3 'cajas' de almacenamiento; por lo tanto, existen ocho 'cajas' donde podemos guardar valores. Realmente estas cajas se corresponden con el modelo abstracto de lista. Los números que queremos ordenar son los siguientes:

N: 3,4,6,3,5

El funcionamiento es sencillo. A cada número lo vamos metiendo en la caja que le corresponde según su propio valor. Así, el número 5 va a la caja 5. Después de esto, tenemos nuestras cajas del siguiente modo:

Caja 0: Vacía	*Caja 1: Vacía*	*Caja 2: Vacía*
Caja 3: '3','3'	*Caja 4: '4'*	*Caja 5: '5'*
Caja 6: '6'	*Caja 7: Vacía*	

Una vez realizado esto, sólo tenemos que recorrer las cajas por orden de numeración, saltando las vacías:

Lista ordenada : '3', '3', '4', '5', '6'.

Ya tenemos nuestra serie de números ordenada. Rápido ¿verdad? El lector estará pensando que ocurre si usamos, por ejemplo, número de 16 bits. ¿Tenemos que usar 2^16 cajas? Por supuesto que no es necesario; se pueden descomponer nuestros números y hacer varias pasadas. Así, por ejemplo, podemos dividir los números de 16 bits en la parte alta de 8 bits y en la parte baja. Usamos 2^8 cajas. En la primera pasada introducimos los números en las cajas en función de los 8 bits bajos, mientras que en la siguiente pasada introducimos los números en función de los 8 bits altos. Importante: en la segunda pasada vamos recorriendo los números en el orden en el que nos los dejó la primera pasada. La idea es sencilla: es como ordenar un conjunto de libros, donde primero hacemos unos montones ordenados por autor y luego se realiza otra ordenación sobre los montones anteriores ordenando, esta vez, por título.

Tenemos que añadir luces a la escena, por lo que el mes que viene trataremos este tema a fondo para conseguir mover luces en nuestro mundo

Podemos obtener aún más velocidad si enlazamos la lista de triángulos con punteros. De modo que en la ordenación no realizamos ni un solo movimiento, ya que sólo modificamos el valor de los punteros.

Veamos todos esto implementado en C++. Para ello, definimos la siguiente estructura correspondiente a un triángulo:

```
BEGIN RXTRI

  VRT vrt[3];           // 3 vértices
  WORD key;             // Z media de la cara
  WORD color;           // Color Flat de la cara
  struct RXTRI *next;   // Siguiente cara. NULL= ultima

END RXTRI;
```

La función que se encarga de ordenar y pintar los triángulos en nuestro ejemplo mediante el algoritmo Radix sort, aparece en el listado 1:
Esto ha sido todo por este mes. En el Cd-Rom que acompaña a la revista se incluye el ejemplo correspondiente al tema. Nuestro visor va creciendo mes a mes y, poco a poco, vamos obteniendo imágenes cada vez más realistas. Las imágenes de este mes son, sin duda, mucho más vistosas que las del mes pasado en alámbrico. Pero todavía queda mucho por avanzar. Tenemos que añadir luces a la escena, por lo que el mes que viene trataremos este tema a fondo para conseguir mover luces en nuestro mundo. La dirección electrónica del autor sigue disponible para que el lector envíe quejas, sugerencias, ideas, preguntas o código. Hasta el próximo artículo.

Jesús de Santos García
icg_ent@hotmail.com

LISTADO 1

```
// ORDENACION POR RADIX [ RADIX SORT ]

#define BITS   8
#define RADIX  256     // radix = 2 ^ bits
#define PASS   2       // 2 ^ (bits * pass ) = bits totales

// Cajas y cabeceras de las cajas.

RXTRI **pgroups[RADIX], *groups[RADIX];

void radix_draw(GFXMODE_EX *gfx, camera &cam) {

   SDWORD i,j;
   WORD val;
   RXTRI *cur, *first, **pfirst;

   first=tri_buffer;
   if(cur_tri==0) first=NULL;

   for(i=0;i<PASS;i++) {

   // Reiniciar cada lista

      for(j=0;j<RADIX;j++) pgroups[j]=groups+j;

   // Almacenar cada triángulo en la lista correspondiente

      for(cur=first; cur; cur=cur->next) {

      val = (WORD)(((cur->key) >> (i*BITS)) & (RADIX - 1));
         *pgroups[val]=cur;
         pgroups[val]=&cur->next;
      }

   // Terminar cada lista

      for(j=0;j<RADIX;j++) *pgroups[j]=NULL;

      pfirst=&first;

      for(j=RADIX-1;j>=0;j—) {

         if(groups[j]) {

            *pfirst=groups[j];
            pfirst=pgroups[j];

         }

      }

   }

   // Pintar

   for(cur=first; cur; cur=cur->next) {

      triangle_flat(cur->vrt,3,cam.buffer,gfx,cur->color);

   }

}
```

Autor: Antonio Marchal

Pilas, colas y otros datos

Este mes vamos a ver varios tipos de datos, que aunque no son tenidos en cuenta por DIV Games Studio, son fácilmente conseguibles, mediante diferentes técnicas de programación. Las colas, las pilas o los datos con tamaño indeterminado serán algunos de los conceptos de los que hablaremos en este artículo. Esperamos que os resulte útil, y sin más preámbulos empecemos.

Antes de comenzar a describir los nuevos datos vamos a ver qué tenemos ya creado. Es decir, vamos a realizar un inventario de las herramientas de que disponemos para la construcción de estos nuevos datos. Luego veremos una pequeña descripción de cada tipo de dato nuevo, así como un ejemplo práctico, para poder ver todas las características de su utilización. Pasemos a ver las herramientas de que disponemos para trabajar.

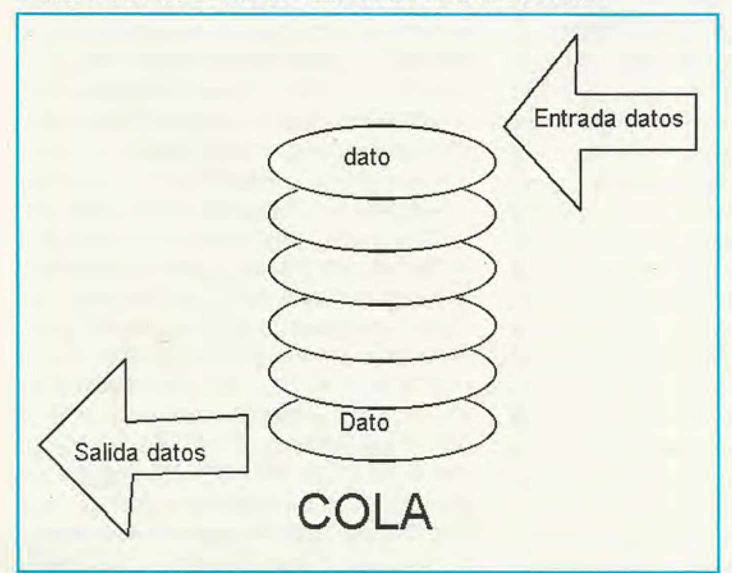
LAS HERRAMIENTAS

Dentro de DIV Games Studio, se podría decir que únicamente existe un tipo de datos. Éste tiene un rango determinado y puede contener un valor que determine un número o un texto. Dejando esta clasificación aparte, nos centraremos en la unión de estas variables. Esto ocurre cuando usamos las tablas y estructuras, que serán muy útiles para nuestros fines.
Las tablas son un compendio de variables que suelen guardar valores relacionados, y que son accesibles a través de un número de orden, que es el de su posición. Las tablas también pueden ser usadas como punteros, pero éste no será nuestro caso. Únicamente accederemos a las mismas, usando su número de orden, indicándolo entre corchetes. Si se quiere más información a cerca de las tablas se puede consultar la documentación que incluye DIV Games Studio.

Las tablas son un compendio de variables que suelen guardar valores relacionados, accesibles a través de un número de orden

En cuanto a las estructuras la cosa cambia, ya que su uso es más complicado. Primero, podemos crear tablas de estructuras, es decir, que podremos crear varios elementos de una determinada estructura. Por otro lado, una de éstas puede contener varios campos, por denominarlos de algún modo. Éstos pueden ser variables, tablas e incluso otras estructuras. Se podría conseguir una estructura de estructuras de estructuras... y así indefinidamente. Aunque su uso parece más complicado, a simple vista, que el de las tablas, a veces el uso de estructuras simplifica el uso de tablas, cuando se tienen varios parámetros para acceder a un dato de un tabla. En estos casos será mejor crear un campo por cada parámetro, y actuar en consecuencia.
Antes de terminar hay que comentar el hecho del acceso a las variables. Esta peculiaridad del lenguaje, unido a la recursividad de los procesos, puede servir para nuestro cometido. Imaginamos que sabréis que hay tres tipos de variables: globales, locales y privadas. De estos tres tipos, únicamente los dos últimos nos serán útiles. El truco está en que cada proceso tiene su propia variable, bien local, bien privada. Es decir, que cada vez que llamemos a un proceso éste tendrá sus propias variables, que serán, por así decirlo, internas. Sabiendo esto, y además contando con la recursividad de los procesos. Entendiendo como recursividad a la posibilidad de que un proceso se llame a sí mismo. Podemos conseguir, que una determinada variable vaya cambiando, a cada llamada a un proceso. Resumiendo, podemos conseguir que cada proceso lleve una información inherente, que condicione su funcionamiento. Dicho en términos sencillos, sería lo mismo que decir, que cada "bicho" tiene su "ficha" para saber qué hacer en cada momento.
Vistas todas las herramientas vamos a pasar a crear nuevos datos, comenzando con las pilas, viendo su definición y uso. Esto lo haremos con los otros tipos de datos. Pero no nos enrollemos más, y pongámonos manos a la obra.

SEGURAMENTE A MAS DE UNO OS HABRA TOCADO ESPERAR EN UNA COLA.



LAS PILAS

Cuando se quiere definir qué es una cola, muchas veces se recurre al mismo ejemplo. Imaginemos un bar donde tienen roto el lavavajillas. El camarero deberá lavar todos los platos, pero ¿qué sistema seguirá? Pues el que denominamos pila. Éste consiste en que el último en entrar es el primero en salir. Si imaginamos una pila de platos, de ahí su nombre, entenderemos el funcionamiento rápidamente.
Pero esto, dicho así, es muy bonito pero no dice nada. Debemos estudiar el verdadero funcionamiento de una pila, y qué debemos hacer para conseguir trasladarla al lenguaje de DIV Games Studio. Se podrían dar dos pasos : uno cuando se mete un elemento, en nuestro ejemplo un plato nuevo a lavar que llega a la pila ; y otro cuando se coge un elemento, en nuestro ejemplo el plato que cogemos de la pila.

Bits y banderas

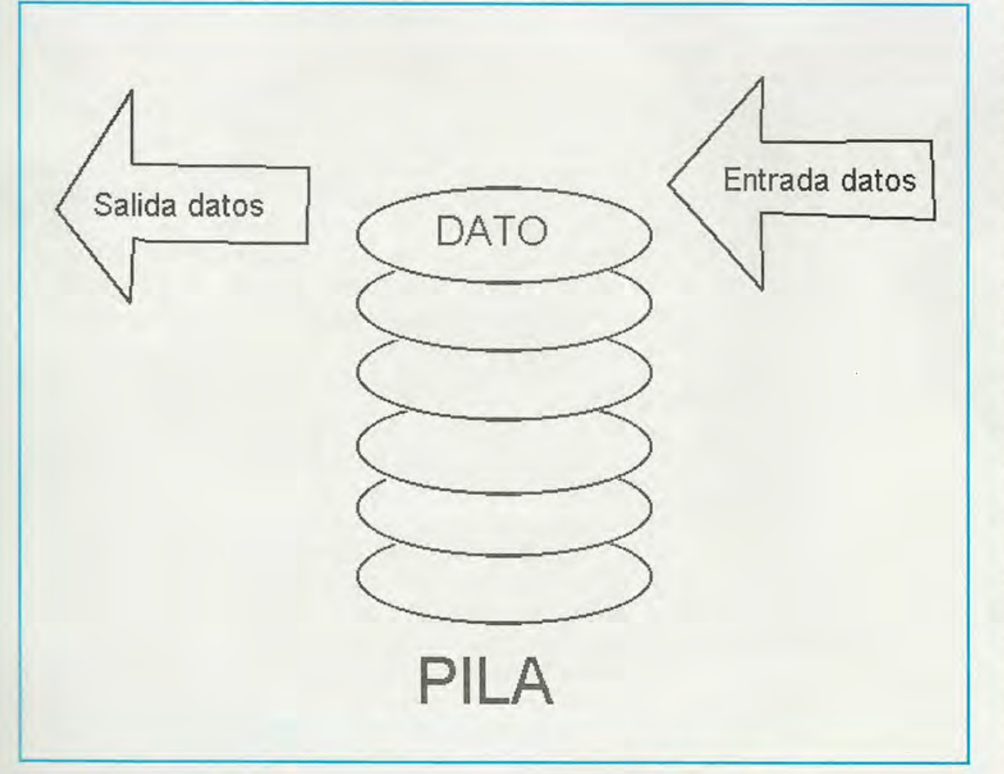
Existen otros tipos de datos que usan algunos lenguajes que únicamente tienen dos valores, o estados, como también se les denomina. Tener agrupadas varias de este tipo de variables a nivel de bit también se pueden conseguir con DIV Games Studio, ya que existen operadores para trabajar a nivel de bit, como pueden ser el AND y el OR. Esto es así porque el lenguaje mantiene cierta relación con otros que también lo usan. La forma de funcionar es igual que estos otros lenguajes.

El primer problema con el que nos encontramos es que con DIV no se pueden crear datos nuevos. Esto nos condiciona a que nuestra pila tenga un número determinado de datos. Para empezar realizaremos una tabla, que será donde crearemos nuestra pila, del tamaño máximo de la misma. También tendremos una variable que nos indicará el tamaño actual de la pila, es decir, el número de elementos contenido en cada momento. Con esto, podremos ya crear nuestros dos procesos, uno que meta datos en la pila y otro que los saque.
La pila es, con este tipo de estructura, la más fácil de manejar ya que, para introducir un dato, lo tenemos que hacer en la posición del número de elementos. Es decir, si hay cero elementos, en la primera posición, si hay 1, en la segunda, etc. Pero como las tablas comienzan con el valor cero, únicamente debemos indicar el número de elementos para introducir el dato en la posición correcta. Después de esto únicamente tendríamos que incrementar el valor que guarda el número de elementos, para tener actualizada nuestra pila. Cuando queramos sacar un dato debemos hacerlo también de la posición final de la tabla, que está una posición antes de la designada por el número de elementos. Si quisiéramos eliminar un elemento bastaría con decrementar el número de elementos, pudiendo poner un 0, en la posición vacía. Únicamente deberemos tener cuidado con los límites de la tabla, tanto por el principio como por el final, para no salirnos, y hacer una asignación incorrecta. Salvo estas comprobaciones, la pila funcionará perfectamente sin otras técnicas de programación.

El primer problema con que nos encontramos es que, con DIV, no se pueden crear datos nuevos

Para mucha gente que tenga experiencia con otros lenguajes que no sean DIV Games Studio, a lo mejor les habrá chocado un poco el funcionamiento de esta pila ya que, normalmente, las pilas suelen funcionar hacia abajo. Es decir, se toma una posición, y en vez de contar los elementos de forma positiva, se toman de forma negativa. El uso que hacemos aquí de las pilas tiene el mismo fundamento que el de las pilas de otros lenguajes, por lo que se podría dar por válido. También decir que existe una DLL, creada por un programador DIV, que maneja este tipo de datos, pero creando nuevos. Si no queremos complicarnos la vida mucho siempre podemos usar el método más afín con nuestras preferencias.

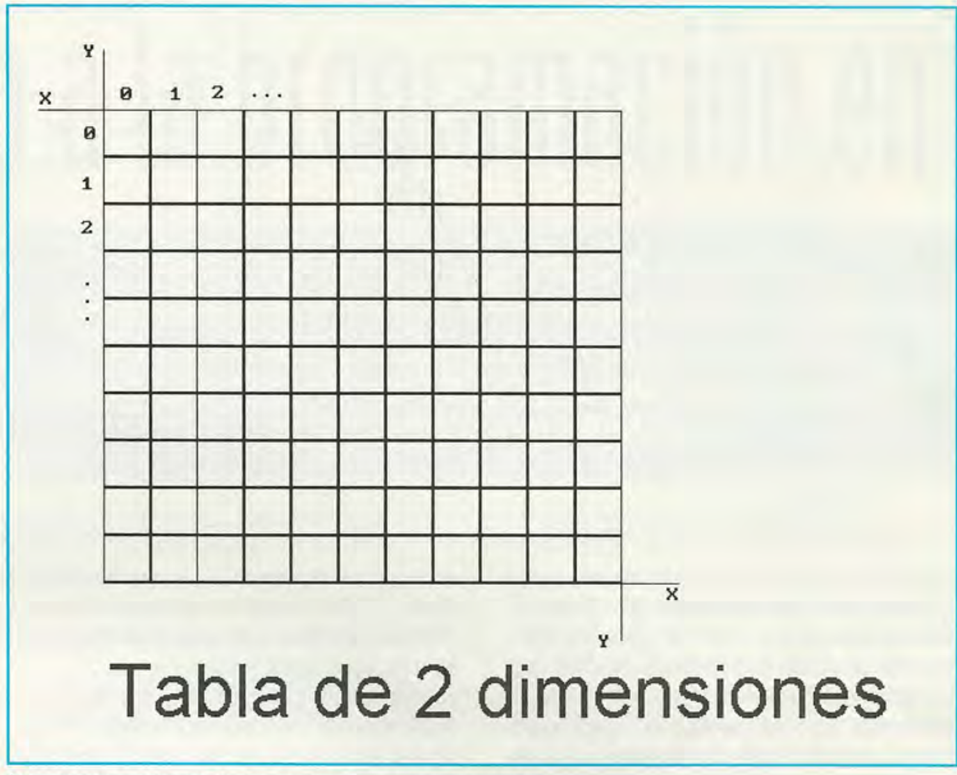
POR SI NO PODEIS IMAGINAR A UN CAMARERO, PONDREMOS UN GRAFICO.



LAS COLAS

Siguiendo con los ejemplos típicos, siempre que se habla de una cola se imagina la de un cine o un teatro a la hora de sacar las entradas. En este ejemplo, a no ser que alguien se cuele, el primero en llegar será el primero en salir. La diferencia con la pila es obvia, mientras en ésta puede haber un dato con una "antigüedad" muy superior a la de los demás, en una cola, todos tendrán la misma antigüedad ya que el primero que se va es siempre el que más lleva esperando.
Como ocurría en el caso de las pilas, debemos indicar el funcionamiento con DIV Games Studio, para lo que crearemos los dos mismos procesos, uno que introduzca datos y otro que los saque. Esta vez también tendremos una tabla y una variable que cuente los elementos, pero los procesos a realizar serán distintos.
Cuando queramos meter un dato actuaremos como la vez anterior, posicionándolo al final de la tabla, es decir, exactamente detrás del último elemento creado. Se usarían instrucciones muy parecidas a las usadas con la pila. Introducimos el valor e incrementamos el contador de elementos, para indicar que hay uno más. Pero a la hora de sacar los datos la cosa cambia, ya que no podremos hacerlo como antes.
En el caso de las pilas se cogía el final de tabla y se operaba con ella. Se introducía un elemento o se sacaba de este lugar, sin interferir con los demás elementos. Sólo debíamos conocer el lugar donde estaba posicionado el dato en cuestión. Pero cuando usamos una cola la cosa cambia, por lo menos a la hora de sacar datos.
Imaginemos la cola del cine, comentada como ejemplo, cuando llega alguien nuevo se pone al final, y no modifica en nada a los que estén haciendo los demás. Pero cuando alguien compra su entrada y se marcha deja su lugar libre, por lo que todos los demás deben avanzar un puesto en la cola. Con los datos pasa lo mismo, y cuando queramos sacar un dato lo haremos de la posición 0 de la tabla, para luego mover todos los elementos una posición adelante. No debemos olvidar contabilizar la salida del elemento, para tener el número de éstos actualizado.

Para mover los elementos lo podemos hacer mediante un bucle que recorra la tabla, colocando el valor numerado como 1 en la posición 0, el segundo elemento como el primero, el tercero como el cuarto y así hasta que hayamos movido todos los elementos. Debido a la obligatoriedad de este movimiento, ocurre que las colas son más lentas que las pilas, por lo menos usando esta implementación. También decir que hemos de tener un control sobre el final y el principio de la tabla, para no salirnos por ningún lado. Aunque esta vez únicamente es importante el final, ya que si no hay elementos en la cola lo único que pasa es que los datos recogidos son incorrectos, pero no realizamos en ningún momento un acceso ilegal a la tabla.
Como ocurría con las pilas, Almi, el programador del que he hablado antes, ha creado una DLL que permite el manejo de este tipo de datos, y que podemos encontrar en Internet, dentro de la página oficial de DIV Games Studio. Como se ha dicho anteriormente, estas librerías crean nuevos datos, y son realmente pilas y colas, más "puras", que las comentadas en estas líneas.



AQUI VEMOS LA ESTRUCTURA DE UNA TABLA DE DOS DIMENSIONES.

TROZOS DE DATOS

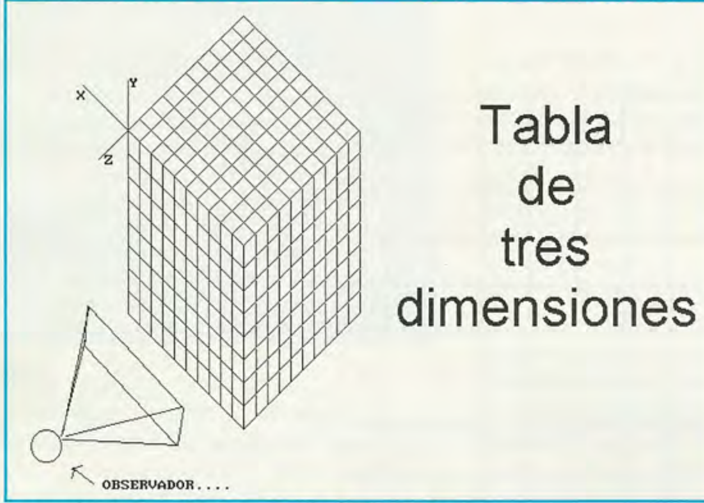
Existe a veces la necesidad de usar datos que son distintos. Aunque podemos usar estructuras, que agrupan datos de distinta índole, todos ellos son completos. Es decir, todas las estructuras que conformen una tabla de las mismas tendrán el mismo número de campos, por lo tanto la misma longitud.
Pero puede darse el caso de que lo que queramos sean trozos de datos. Es decir, que cada dato completo tenga distinta longitud. Por ejemplo, podemos necesitar guardar distintas habitaciones, en cada una de ellas puede haber un número de elementos distintos. Aquí el uso de datos de distintos tamaños parece ser la solución. Esto lo podemos conseguir también usando trucos, con DIV Games Studio.
Para llevarlo a la práctica usaremos dos tablas: la primera de ellas será un índice, y la otra la que almacenará los datos en sí. Es decir, en esta última tabla es donde estarán todos los datos, unos seguidos de otros. Por lo tanto, debe tener un tamaño suficiente para poder contenerlos a todos. Solamente debemos saber dónde comienza cada dato y su longitud. Estos dos valores los guardaremos dentro de la tabla que haga de índice.
Por último, únicamente debemos saber el número de elemento a leer, buscaremos dónde empieza dentro de la tabla índice, sabiendo que tenemos que multiplicar por dos. Luego, mirando la longitud, cogeremos el dato en su totalidad. De esta manera, el primer dato puede tener 5 elementos, el segundo 2, el tercero 4, etc. Y podremos saber dónde comienza cada uno automáticamente.
Existen otros métodos para usar este tipo de datos. Uno de ellos se basa en la inclusión de un código de control. Éste debe ser un valor imposible. Es decir, que sea posible, que esté contenido dentro de la tabla como un dato normal. Podemos usar números negativos o muy grandes para conseguir esta "imposibilidad". Después debemos crear una tabla donde guardar todos los datos y además, entre cada uno de ellos, un código de control que los delimite. Tenemos que pensar que la tabla debe tener un espacio suficiente para almacenar toda esta información.
Finalmente, para leer un dato recorreremos toda la tabla, comprobando si nos encontramos un código de control, para así conocer dónde empiezan y acaban los datos y, por tanto, poder contarlos. Para ir a un dato determinado deberemos saltarnos todos los datos anteriores, hasta llegar al que queramos. Este último método tiene la inconveniencia de la velocidad, ya que debemos recorrer la tabla hasta encontrar el dato requerido. Y si, por un casual, está al final de la tabla, requerirá un tiempo de ejecución y proceso bastante alto. Por otro lado, para reducir esta espera, podemos usar otros trucos, junto con éste, como el de comenzar por el final de la tabla cuando este dato esté más cerca de esta posición que del principio.
Si quisiéramos implementar este tipo de datos con los antes comentados, es decir, las pilas y las colas, sería más adecuado usar el último método descrito, ya que no depende de dos tablas sino de una. Aunque este juicio, a priori, puede no resultar muy apto para mucha gente, debido a que es más complicado que el anterior.

2 DIMENSIONES, 3 DIMENSIONES, 4 DIMENSIONES...

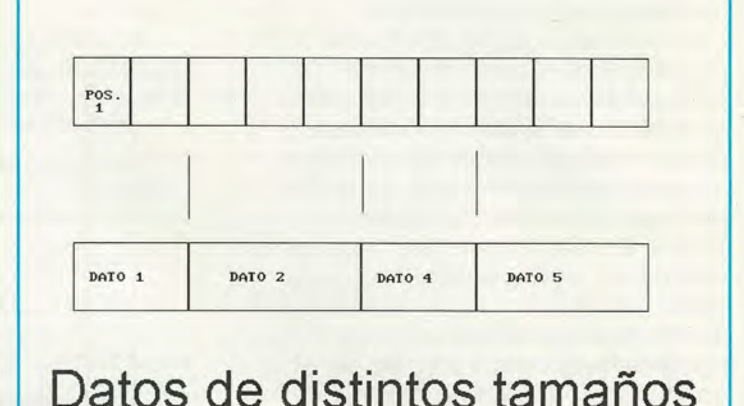
Uno de los problemas que más ha acusado la gente que no tenía mucha experiencia en programación anterior a DIV es el hecho de crear tablas de más de una dimensión. Aparte, en otros lenguajes sí existe esta posibilidad, y si no, se utilizan trucos, como siempre, para simular en todo lo posible la misma.

Tileado

Si queremos usar alguna técnica de tileado, es decir, construir un mapa usando trozos de otros gráficos y repitiéndolos para conformar un nuevo gráfico, usaremos tablas de dos dimensiones ; y si queremos guardar además otros datos, como puede ser la fase, o incluso todos los mapas que haya en un juego - si hay más de uno -, y además que todo esté dentro de una variable, podemos usar una tabla de 3 dimensiones. Entonces debemos emplear el método descrito para convertir las tablas DIV a 2 ó 3 dimensiones. También es conveniente crearse un tileador o programa mapeador para facilitarnos el trabajo.



El cubo de Rubik... No, una tabla de tres dimensiones.



Con esta imagen comprobamos como poder guardar datos "diferentes".

Pero para mucha gente lo de las dimensiones le puede sonar un tanto a chino, por lo que convendría explicarlo. Normalmente, cuando creamos una tabla sólo podemos designar que la misma tenga un número dado de elementos. Pero imaginemos que tenemos un mapa, con coordenadas horizontales y verticales. Y que dividimos todo el mapa en cuadritos, y ordenamos todo el mapa refiriéndonos a cada trocito del mismo por su posición en el susodicho, dada dicha posición por la combinación de las dos coordenadas.
Si quisiéramos codificar este mapa y guardarlo como una variable necesitaríamos manejar dos dimensiones, serían las dos coordenadas. Pero como hemos dicho antes, DIV Games Studio, exclusivamente permite usar una. Esto tiene solución.
Lo que haremos será crear una variable que sea el producto de las dos dimensiones, y almacenaremos los datos en ella. Es decir, crearemos una tabla cuyo tamaño total será el del ancho del mapa multiplicado por el alto. Con esto conseguiremos poder codificar nuestro mapa y tenerlo guardado en una variable.
Pero luego debemos acceder al mismo, por lo que deberemos usar una fórmula. Imaginando que al ordenar el mapa hemos empezado por la esquina superior derecha, y que hemos seguido en el orden elegido, para la izquierda, hasta llegar al final de la línea, y luego para abajo, y que en "x" se guarda la coordenada horizontal y en "y" la vertical, ya nos pondríamos a crear la fórmula. Ésta devolvería la posición dentro de la tabla creada, donde se encuentra el valor requerido, y sería algo así:

```
posición=x+(y*ancho);
```

Para usar esta fórmula únicamente debemos saber el ancho del mapa, pudiendo conseguir un mapa tan alto como lo permita la tabla usada. También se puede conseguir el mismo efecto mediante una estructura, pero su uso sería algo más complicado.
En este caso, si usáramos una estructura, la declaración de la misma podría ser la siguiente:

```
STRUCT mapay[alto];
        mapax[ancho];
END
```

Esta estructura tendría la misma organización que la usada para la tabla, y para acceder a cualquier elemento usaríamos una sentencia:

```
mapay[y].mapax[x]=valor;
```

Con lo que tendríamos las dos dimesiones arregladas, pero podemos ir mas allá.
Imaginemos un mundo en 3 dimensiones, es decir, un mapa como el anterior, pero que además tiene una nueva variable que indica la profundidad, y a la que llamaremos "z".
Para manejar dicho mundo 3D y guardarlo en una tabla, debemos crearla usando un método análogo al anterior, es decir, multiplicando el alto por el ancho y por el largo, dando así el número de elementos. Para acceder a cualquier dato usaríamos una fórmula como la siguiente:

```
posición=x+(y*ancho)+(z*anho*alto);
```

En este caso, la tabla también estaría organizada como en los casos anteriores, pero cuando se llegara al final del plano continuaríamos con el siguiente plano colocado a una profundidad mayor, y así hasta el fondo.
Si usáramos estructuras, lo que debemos hacer es introducir la estructura anterior en otra anidándolas. La declaración de la misma quedaría así:

```
STRUCT mapaz[largo];
        STRUCT mapay[alto];
                mapax[ancho];
        END
END
```

Y usándose de modo similar al anteriormente descrito, con la novedad del nuevo índice de estructura, el de mapaz, que nos permite designar la profundidad en la cuál nos encontramos. Por lo demás, está organizado como siempre y nos permite almacenar las tres dimensiones sin mayores problemas.
Ahora imaginemos un mundo en 4 dimensiones... o mejor no. Pero imaginemos el caso anterior, y además dividido en habitaciones. En este caso, usaríamos alguna de las técnicas anteriormente descritas, es decir, bien multiplicando por la nueva dimensión, bien anidando una nueva estructura. Así podríamos conseguir todas las dimensiones que quisiéramos.

Despedida y cierre

Bueno, ya nos vamos, hasta el mes que viene, donde continuaremos "DIVirtiéndonos".
Solamente decir que ha finalizado el concurso de DIV, el de la página oficial en Internet. Podéis pasaros a ver los premiados. También, si tenéis alguna duda, podéis dirigiros a mi dirección electrónica, tizo@100mbps.es, donde os contestaré, siempre que pueda, que por suerte es casi siempre. Bueno, nos leemos, y que os Divirtáis programando.

Autor: José Antonio Guerra

Introducción a la programación en Direct X

Miedo. Terror. Pánico. Confusión. Esos y más términos definen el estado del programador de juegos acostumbrado a la simpleza y rapidez del modelo monotarea del entorno MS-DOS cuando se da cuenta de que ha de migrar a Windows.

Un entorno completamente nuevo, multitarea, con un API cuyos manuales de referencia ocupan estanterías..., vamos, que a cualquiera se le quitan las ganas de meterse. Y no hablemos de los gráficos; acostumbrados al cómodo modo 13h, a las normas VESA, ahora nada de esto funciona aquí, todo es distinto. Pero, ¿habrá que volver a reescribir esas rutinas de bajo nivel que tanto sudor nos costaron? La respuesta es NO.

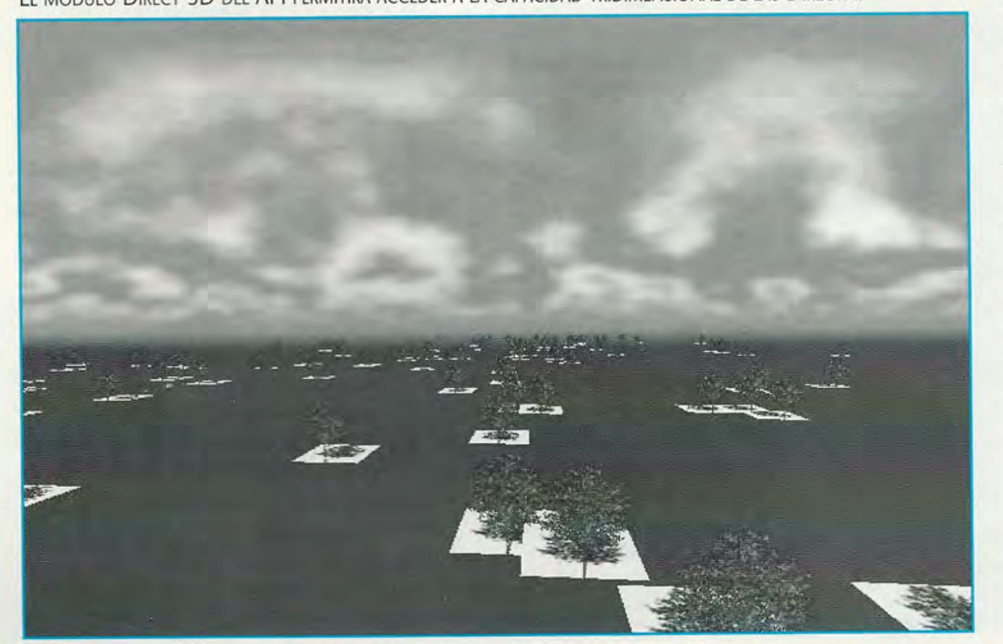
Los chicos de Microsoft, al hacer su sistema operativo Windows 95, se dieron cuenta de que tenían prácticamente todas las áreas del mercado de software cubiertas. Procesadores de texto, hojas de cálculo, programas de autoedición y un largo etcétera estaban en su gran mayoría escritos para el entorno Windows. La razón era obvia: había un interfaz común, un API bastante completo y, sobre todo, una abstracción total de los dispositivos hardware, con lo cual los desarrolladores no tenían que hacer un controlador para cada modelo de impresora o tarjeta de vídeo. Pero se dieron cuenta de que les faltaba un área fundamental: los videojuegos.

La idea fundamental de DirectX es proporcionar un acceso más rápido y directo a estos dispositivos pero sin renunciar a la idea de abstracción del hardware

Ciertamente, las carencias de Windows para programar ese tipo de software eran evidentes; las funciones API de manejo de bitmaps eran lentísimas para estos fines, las funciones de sonido no eran suficientes y se precisaba un acceso más directo a los dispositivos de entrada. De ahí nació una arquitectura específica para solucionar todos esos problemas: DirectX.

La idea fundamental de DirectX es proporcionar un acceso más rápido y directo a estos dispositivos pero sin renunciar a la idea de abstracción del hardware, es decir, que mediante una interfaz única se pueda controlar el dispositivo que haya instalado sin saber nada de su funcionamiento interno. Por ejemplo, esto a los que programen habitualmente en modo 13h les puede traer sin cuidado, ya que dicho modo es un estándar y todos los PC lo soportan, pero cuando hablamos de programar un modo de 16 o 24 bits de color, o peor aún, cuando se trata de vérselas con las tarjetas de sonido (quien lo haya hecho sabrá los dolores de cabeza que conlleva), el modelo de abstracción es una maravilla. Pero, ¿funciona bien dicho modelo?, ¿se podrá acceder libremente a los dispositivos o habrá que ceñirse a un limitado número de funciones? Hablaremos sobre ello más tarde.

El modulo Direct 3D del API permitira acceder a la capacidad tridimensional de las DirectX.



COMPONENTES DIRECTX

La arquitectura DirectX está formada por varios APIs que realizan funciones muy distintas, todas ellas orientadas a la programación de software de entretenimiento:

- **DirectDraw**

Es el API encargado del manejo del dispositivo de vídeo. Permite aprovechar al máximo tarjetas aceleradoras (haciendo *blitting* por hardware, guardando gráficos en memoria de video), así como da soporte a las últimas tecnologías como MMX, AGP y sistemas de varios monitores.

- **DirectSound**

Es el API que controla los dispositivos de sonido digital. Permite mezcla de canales por software y por hardware (en tarjetas de sonido como la AWE 64, por ejemplo), así como el almacenamiento de muestras en la memoria de la tarjeta cuando ésta está disponible. También da soporte a sistemas de audio 3D, permitiendo establecer la reproducción de un sonido en una posición relativa al oyente,

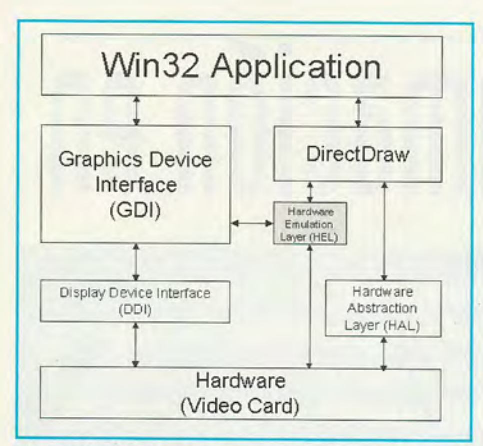


FIGURA 1. ECOMUNICACION ENTRE LA INTERFAZ Y EL DISPOSITIVO EN DIRECTDRAW.

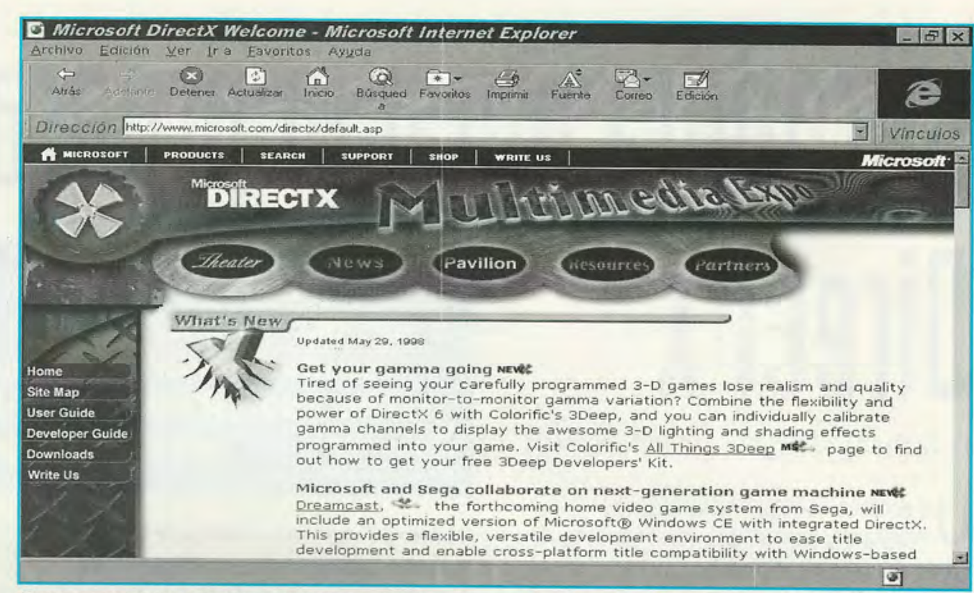
haciendo más realistas juegos como *Quake*, *Tomb Raider* o similares.

- **Direct3D**

Es la extensión de gráficos 3D de DirectX. Posee dos modos de funcionamiento: un modo de alto nivel (*Retained Mode*), en el cual se facilitan clases para manejar modelos, interpoladores, mallas progresivas, que ofrecen una solución rápida para manejar gráficos, y un modo de bajo nivel (*Immediate Mode*) que permite un control total de la renderización, para casos en los que tengamos que usar métodos propios para nuestro motor 3D, como árboles BSP. Por supuesto, ambos métodos permiten sacar el máximo provecho a las aceleradoras 3D de última generación, así como aprovechar las tecnologías como MMX en caso de no poseer una de esas "maravillas".

- **DirectPlay**

Este API facilita enormemente la realización de juegos multijugador. Permite usar indistintamente varios tipos de conexión entre ordenadores, como red IPX, conexión por cable serie o conexión por protocolo TCP/IP, permitiendo el juego a través de Internet o redes locales.



DESDE LA PAGINA WEB DE MICROSOFT PODREMOS ACCEDER A CANTIDAD DE SERVICIOS REFERENTES A DIRECT X.

- **DirectInput**

Es el encargado de manejar los dispositivos de entrada que podremos usar para nuestros juegos. Actualmente da soporte al ratón, al teclado y al joystick, así como a los nuevos controladores de juegos como *force-feedback*, que permiten controlar joysticks que transmiten sensaciones al usuario, como aplicar fuerzas en la palanca, vibraciones en caso de recibir impactos y variar la resistencia del dispositivo al movimiento.

- **DirectSetup**

Es un sencillo API que nos permitirá controlar la instalación de los controladores DirectX.

- **AutoPlay**

Permite la ejecución automática de software en CD-ROM con tan sólo insertarlo en la unidad. Aunque este servicio es nativo de Windows 95, es considerado fundamental en la filosofía de DirectX

ARQUITECTURA DIRECTX

Para comunicarse entre la interfaz y el dispositivo, DirectX dispone de dos elementos: el HAL y el HEL.

El HAL (*Hardware Abstraction Layer* o Capa de Abstracción de Hardware), es llamado para las funciones DirectX que el hardware lleva implementado. Por ejemplo, si una tarjeta de vídeo es capaz de acelerar el dibujado de un bitmap con *alpha blending* o una tarjeta de sonido puede hacer una mezcla de canales



Funciones HEL de DirectDraw

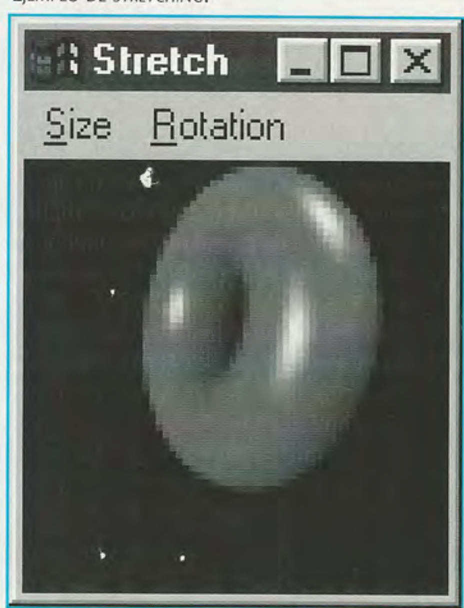
3D
BLT
BLTSTRETCH
PALETTE
COLORKEY
BLTDEPTHFILL
CANCLIP
CANCLIPSTRETCHED

Color Key
SRCBLT

FX
BLTMIRRORLEFTRIGHT
BLTMIRRORUPDOWN
BLTSHRINKX
BLTSHRINKXN
BLTSHRINKY
BLTSHRINKYN
BLTSTRETCHX
BLTSTRETCHXN
BLTSTRETCHY
BLTSTRETCHYN

Surface
FLIP
MIPMAP
OFFSCREENPLAIN
PALETTE
PRIMARYSURFACE
TEXTURE
ZBUFFER

EJEMPLO DE *STRETCHING*.



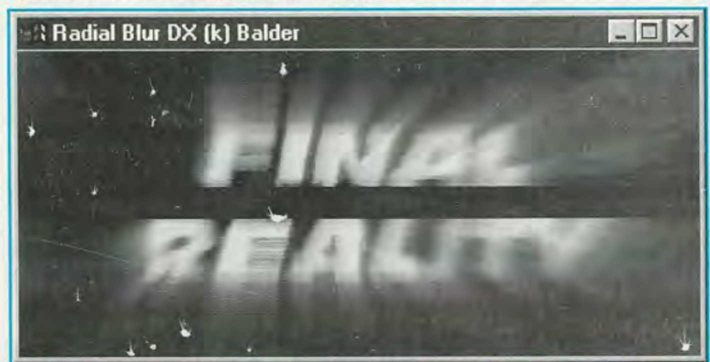
por hardware, el interfaz se comunica a través del HAL directamente con el hardware, aprovechando así la máxima potencia ofrecida por los dispositivos aceleradores y dejando tiempo libre a la CPU para hacer otras cosas.
Por otra parte, el HEL (*Hardware Emulation Layer* o Capa de Emulación de Hardware), es llamado para las funciones DirectX que el hardware no soporta directamente, dando así un soporte bastante completo de funciones. Por ejemplo, si el hardware de vídeo no soporta el dibujado de bitmaps con transparencia, las rutinas del HEL son las encargadas de realizarlo. Gracias a la abstracción, el programador de videojuegos se despreocupa de implementar las funciones que controlan directamente el hardware, aumentando la compatibilidad y disminuyendo el tiempo de desarrollo, a la vez que permite que dispositivos que se fabriquen tras el desarrollo del producto puedan ser totalmente aprovechados por el juego. Pero hay que resaltar algunos puntos buenos y otros no tanto de la implementación de los controladores de DirectX 5.

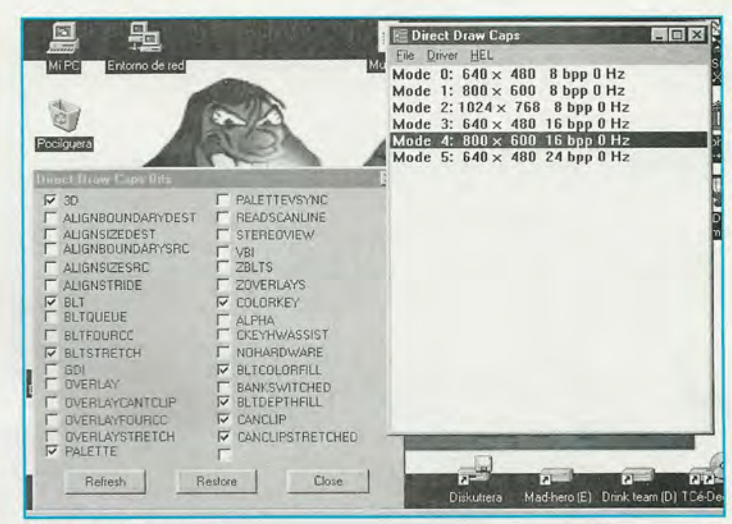
- **Puntos positivos:**
  - Total aprovechamiento de la aceleración por hardware, cuando está disponible.
  - Controladores HEL distintos para procesadores con extensiones MMX o sin ellas, con lo cual, el aprovechamiento de los recursos del procesador es automático.
- **Puntos negativos:**
  - El HEL no soporta todas las funciones del interfaz DirectX, sino sólo las más importantes están presentes.
  - La implementación de los controladores HEL no es tan rápida como cabría esperar, sobre todo la implementación para los procesadores sin MMX.

Evidentemente, el HEL siempre tiene las mismas funciones sea cual sea el dispositivo de vídeo instalado

De todos estos puntos se observa claramente que el máximo rendimiento de DirectX se obtiene en configuraciones con hardware acelerado e incluso con procesadores MMX. En configuraciones con un 486 o Pentium normal y una tarjeta de vídeo poco potente, el rendimiento del HEL en el dibujado de bitmaps es aproximadamente un 20-30% más lento que usando una rutina equivalente en ensamblador, aunque dada la rápida evolución del hardware de vídeo instalado en los PC's domésticos y la compatibilidad y estabilidad de código que nos ofrece el interfaz de DirectX es una pérdida que, en la mayoría de los casos, se compensa. Además, con la posibilidad de acceder directamente al *buffer* de vídeo (tema que veremos en siguientes entregas) se pueden implementar a mano esas rutinas si fuese necesario, e incluso implementar aquellas funciones que el HEL no contemple.

EJEMPLO DE UN EFECTO GRAFICO IMPLEMENTADO EN DIRECTX.





CAPTURA DE *DDCAPS.EXE*.

Un punto muy importante a destacar, y del que no se habla mucho en la documentación original de Microsoft, es el de las funciones soportadas por el HEL de DirectDraw, es decir, las funciones gráficas que son compatibles en todos los ordenadores con DirectX. Es importante respetar esta relación, ya que al utilizar alguna que no se encuentre entre éstas tendremos el problema de que en nuestro juego esa maravillosa rotación que se veía bien con una Virge no aparezca por ningún lado en el ordenador de nuestro vecino. Por ello, en caso de usar alguna de estas características se comprueba si está soportada por el controlador DirectX, y si no lo está usar algún efecto alternativo, o mejor aún, codificarlo nosotros mismos. De todas formas, ahí va la relación de funciones HEL para que el lector la tenga en cuenta y, además, durante el curso se irán advirtiendo excepciones de este tipo para que los programas DirectX sean totalmente ejecutables sobre cualquier configuración.

ESTRUCTURA DEL SDK

La estructura de directorios del SDK de DirectX 5 es la siguiente:

- **\docs**: Contiene la documentación del API en formato Word y en ficheros de ayuda, tanto en inglés como en japonés (por si a alguien le resulta mas cómodo).
- **\foxbear**: Un sencillo juego de plataformas para probar las rutinas 2D de DirectX.
- **\license**: La licencia de distribución de las librerías en tiempo de ejecución (*runtimes*).
- **\media**: Los ficheros de gráficos, sonidos y modelos 3D de los ejemplos.
- **\redist**: Aquí están los *runtime* que deberemos distribuir junto a nuestros programas que utilicen DirectX.
- **\rockem**: Un juego de lucha 3D usando DirectX. Muestra las capacidades de 3D y sonido de DirectX.
- **\sdk**: Aquí empieza lo bueno; veamos los subdirectorios:
- ***\sdk\bin***: Aquí están todas las utilidades del SDK junto a casi todos los ejemplos ya compilados. En este directorio no están sus ficheros de datos, así que deberéis copiarlos desde el directorio *media*. De las herramientas hablaremos mas tarde.
- ***\sdk\inc***: Los ficheros *include* de DirectX
- ***\sdk\lib:*** Las librerías de DirectX para los compiladores Microsoft Visual C, Borland y Watcom.
- ***\sdk\samples***: Los códigos fuente de todos los ejemplos del SDK. Aquí hay cosas que no vienen en la documentación, así que ya las iremos contando.

HERRAMIENTAS DEL SDK

En cuanto a las herramientas del SDK son las siguientes:



TUNEL CON DIRECT3D.

Ddcaps.exe: Nos permite visualizar rápidamente las capacidades del HEL y del HAL de DirectDraw. Evidentemente, el HEL siempre tiene las mismas funciones sea cual sea el dispositivo de vídeo instalado, aunque variará dependiendo de la versión de DirectX instalada. Este punto hace que esta utilidad sea imprescindible para ver qué métodos se pueden usar sin miedo a que las aplicaciones que hagamos sean incompatibles entre equipos con distintos adaptadores.
Ddtest.exe: Se trata de una completa herramienta que nos permite probar cualquier método de DirectDraw. Así, podremos descubrir errores aparentemente correctos en nuestras aplicaciones y practicar de una manera rápida lo que vayamos aprendiendo sin tener que hacer una sola línea de código. Aparte, también dispone de unas sencillas pruebas de velocidad (*benchmarks*).
Ds3dview.exe: Muestra las capacidades de sonido 3D de DirectSound de una manera visual, pudiendo cargar modelos 3D y hacer que estos representen sonidos. Además, permite modificar numerosas variables en tiempo real.
Dsshow.exe: Es una demostración de las capacidades de mezcla de sonidos de DirectSound.
Dsshow3d.exe: Es como el anterior, pero permite modificar las variables que definen la posición, etc., del sonido 3D.
Dsstream.exe: Demuestra la reproducción de *streams* de sonido con DirectSound. Permite reproducir ficheros de sonido grandes que no caben en memoria, como música de fondo almacenada en formato digital, por ejemplo.
Dxview.exe: Esta sencilla utilidad nos permitirá ver los controladores asociados a cada API de DirectX.
Fdfilter.exe: Muestra las capacidades de captura y reproducción simultánea de sonido con DirectSound.
Memtime.exe: Es un pequeño *benchmark* que mide las tasas de transferencia de memoria con el procesador y algunas de las funciones más usadas de DirectDraw.
Mstream.exe: Muestra cómo incluir música en formato MIDI en nuestros programas DirectX.
Viewer.exe: Se trata de un visor de modelos 3D usando Direct3D. Permite ver modelos con distintos tipos de sombreados, focos de luz dinámicos y distintos tipos de filtros. Además, posibilita la visión de mallas progresivas, que básicamente son modelos con un número distinto de polígonos dependiendo del nivel de detalle (son los ficheros que acaban en *_pm*; para variar el nivel de detalle, sólo hay que manipular el control *scroll* que aparecerá a la derecha de la ventana tras cargar alguno de estos modelos).

EL CURSO DE DIRECTX

En este curso de DirectX nos familiarizaremos con el API realizando una práctica dirigida, en la que se tocarán todos los puntos necesarios para poder realizar nuestros juegos bajo este soporte. Durante el tiempo que dure el curso, se hará un juego simple estilo "comecocos", ya que las reglas son conocidas por todos, lo que permitirá ilustrar técnicas de juegos 2D como *scrolls*, mezcla de sonidos y técnicas de animación como el doble o el triple buffer. Durante este tiempo, se van a analizar las API DirectDraw, DirectSound y DirectInput, que son las fundamentales para, posteriormente, en un futuro curso o por cuenta propia de cada uno, aprender el manejo de DirectPlay y Direct3D. Además, durante estos meses iremos viendo detalles y características que no aparecen, o bien están poco documentadas en los manuales oficiales de Microsoft. En el curso usaremos C++ para los programas de ejemplo, usando como compilador el Visual C++ 5.0 de Microsoft, aunque no habrá muchos problemas para compilar los listados usando versiones anteriores de este compilador, o bien utilizando Borland C++ o Watcom. Para todos los que la programación orientada a objetos (POO) no sea lo suyo, pueden estar tranquilos, ya que usaremos poco ese modelo. Tan sólo aprovecharemos la encapsulación para dar más orden y legibilidad a los listados y, sobre todo, porque hay que tener en cuenta que los APIs de DirectX son objetos, aunque sean manejables desde C puro.
Bueno, pues con esto acabamos esta pequeña introducción a la arquitectura DirectX, que empezaremos a manejar a partir del próximo mes. Hasta entonces, si queréis hacer alguna sugerencia podéis mandar un mensaje a *balder@mindless.com.*

TEXTURA CON EL LOGO DE DIRECTX.



Autor: Leticia Krahe

IRC y los videojuegos

En el artículo de este mes, vamos a ver qué influencia tiene el sistema de chat online más extendido del mundo, en el ámbito del desarrollo de videojuegos y en todos los aspectos paralelos a éste, tales como la difusión, la comparación y la propagación de este fenómeno.

Dentro del IRC existen muchísimos nodos de charla, pero nosotros nos vamos a centrar, en esta primera ocasión, en el canal hispano.org de Arrakis uno de los mas transitados por los aficionados al videojuego. También hemos seleccionado este nodo por ser uno de los más utilizados y difundidos, a nuestro juicio, y uno de los de mayor proyección y crecimiento dentro del ámbito del IRC.

MAS EN DETALLE

Dentro de Arrakis podremos encontrar bastantes canales en los cuales el principal tema de conversación es el de los videojuegos, su mundo y más en concreto, su desarrollo. Al igual que para gustos están los colores, para adictos están los canales del IRC.

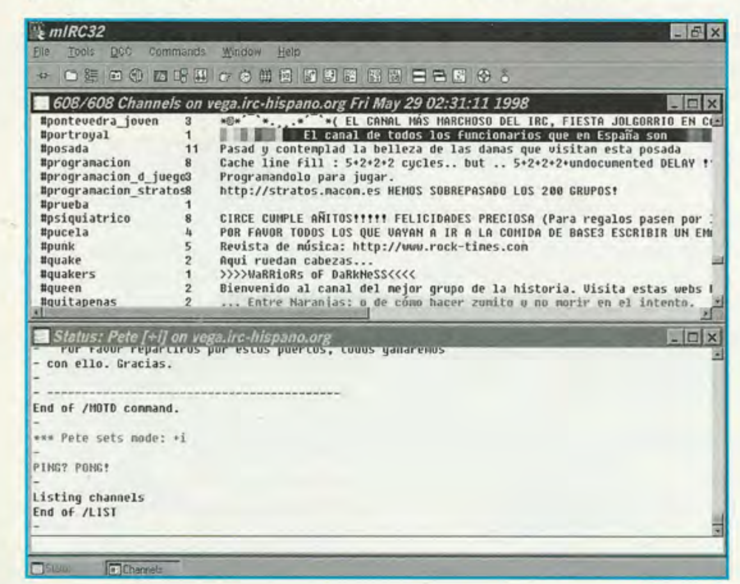
Antes de continuar hay que comentar que en las conversaciones que campean por el canal consigues, desde críticas sin ningún tipo de censura sobre juegos de actualidad, herramientas, técnicas de programación, ..., hasta el intercambio de trucos, cotilleos del sector, o intercambio y venta de juegos usados.

En el canal #PROGRAMACION podemos encontrar un enfoque más variado del tema de la programación, no tan centrado en el mundo del videojuego

Pero más en la línea de lo que a nosotros nos interesa existen canales especializados en juegos PC. Entre los más importantes nos encontramos con tres más que destacan: #PROGRAMACION, #PROGRAMACION_STRATOS e #INFORMATICOS.

PROGRAMACION

Sin lugar a dudas, el canal más candente es el de #PROGRAMACION_ STRATOS, en el cual, dada su naturaleza y el apoyo de la asociación de desarrolladores STRATOS, se da habitualmente una afluencia masiva de programadores, grafistas, músicos y un sinfín de desarrolladores.



Como temas de charla e interés, os podéis imaginar, cualquier apartado dentro del desarrollo de videojuegos es tratado por más de quince usuarios por norma general. Este canal es una buena forma tanto de darse a conocer como de conocer a gente interesante relacionada con este ámbito.

El canal está administrado por #Stratos, Antonio Arteaga, el coordinador de esta asociación, un habitual de nuestras páginas de noticias, caracterizado por su afán de buscar lo mejor para el desarrollo, que ayudará y se encargará de poner en contacto a la gente que haga falta para potenciar la industria del videojuego en España.

En #PROGRAMACION_STRATOS todo vale, y todos tienen cabida, desde el *demoero* más avanzado, al desarrollador iniciado o profesional. Una verdadera joya para el aficcionado y para el experto. Además, como "posdata", decir que si algún usuario estuviera interesado en saber cómo funciona todo esto, no le podemos dar mejor recomendación que la de introducirse en este canal y comentar sus dudas.

En el canal #PROGRAMACION, podemos encontrar un enfoque más variado del tema de la programación, no tan centrado en el mundo del videojuego y más genérico. En él se darán confluencia programadores de cualquier tipo y categoría, pero sin tener por qué estar relacionados, como ya decíamos antes, con el mundo del desarrollo de software lúdico. Alguien que tiene un problema con su compilador del Pascal o una duda con Visual Basic, seguro que encuentra solución a sus interrogantes o problemas. Pero no desesperéis que también podéis entrar en el canal #PROGRAMACION_DE _VIDEOJUEGOS, y éste si que es sólo de videojuegos.

Por último, y como tercer canal más significativo dentro de la programación, podemos hablar de #INFORMATICOS, siendo sin lugar a

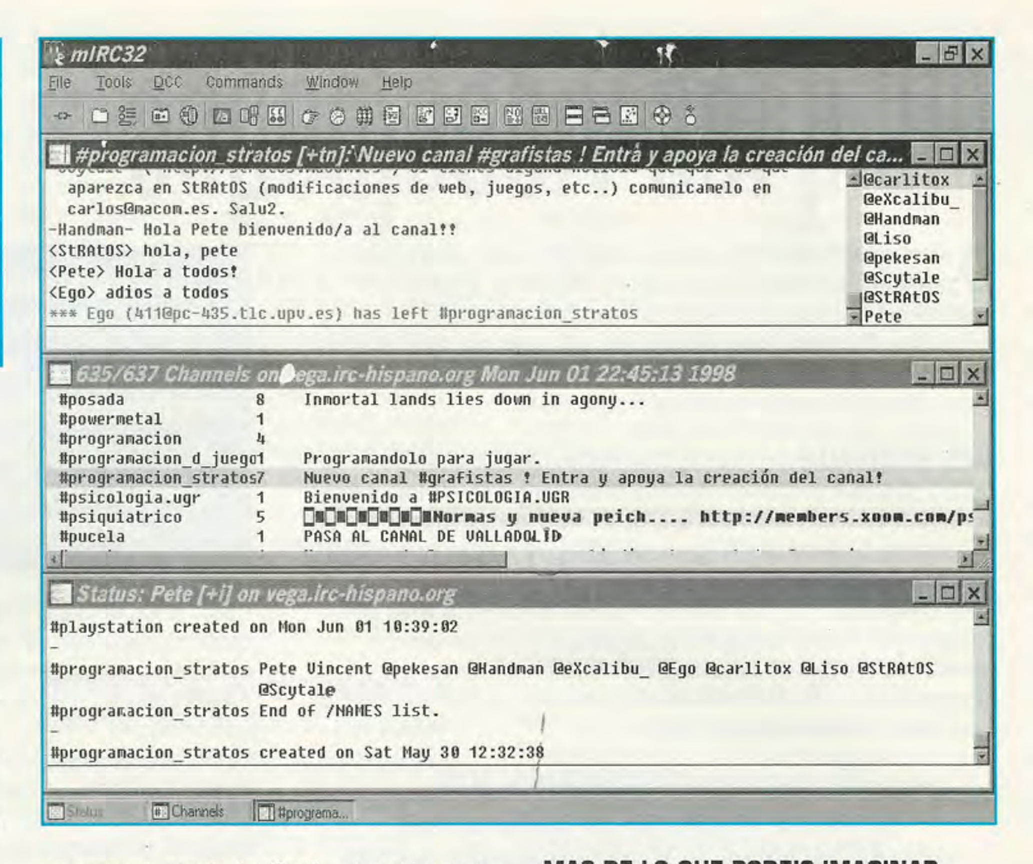
También para jugones

Si aparte de gustarte el desarrollo de juegos, te gusta jugar con ellos, en IRC también encontrarás canales dedicados a tus juegos favoritos. Aquí tenéis una lista de los más conocidos :

#4QUAKE	AQUÍ CONTACTARÉIS CON LA LIGA ESPAÑOLA DE QUAKE
#4QUAKE2	
#CARMAGEDDON	PARA LOS AMIGOS DE LAS LINDAS ANCIANITAS
#MONKEY-ISLAND	ERES UN AVENTURERO, PUES ÉSTE ES TU CANAL
#MERISTATION	LA MEJOR REVISTA ONLINE PARA JUGONES

Canal DIV Games

Si te mola la programación de videojuegos, pero aún no te consideras un auténtico coder, NO PROBLEMO. Únete a la legión de seguidores de DIV, y aprende a programar juegos. Discute tus dudas con el resto de programadores de DIV, intercambia conocimiento y camina por la senda de la programación del videojuego con buen pie. Entra en el canal de IRC dedicado a DIV (canal #DIV) o visita cualquiera de las paginas web : www.redbcn.com/canaldiv o www.divgames.com.



dudas el más genérico de los vistos hasta ahora, siendo más que otra cosa un canal de tertulia variada para aquél que se autodenomine como el nombre del canal.

GRAFISMO

Dentro del desarrollo de videojuegos, no sólo puede interesar la programación, vamos ahora con el apartado visual y con todos aquellos canales que se pueden ocupar del tema. En primer lugar podemos hablar de #DISEÑO, un canal en donde se dan cita verdaderos expertos, por lo general, dentro del campo del diseño, casi siempre orientado al manejo de herramientas como PhotoShop, Fractal Painter, 3Dstudio o similares.

Destacamos un canal de reciente aparición: #GRAFISTAS, de los creadores de #PROGRAMACION_STRATOS

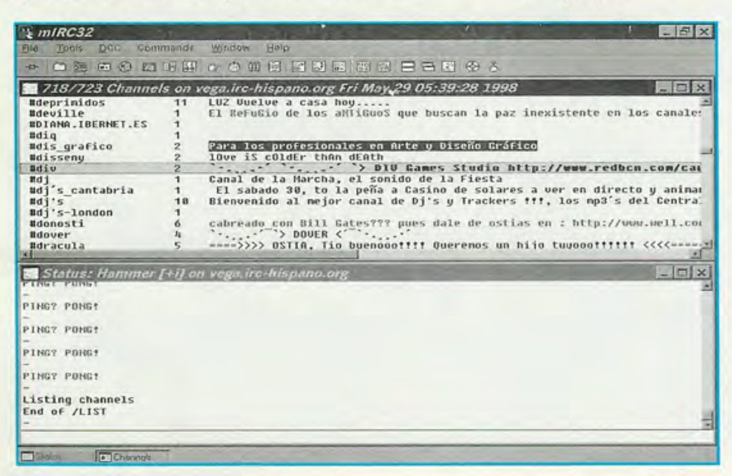
#DISEÑO es un buen sitio para solventar dudas y conocer a gente dedicada en cuerpo y alma a este tipo de menesteres. ¿Qué es un programador sin un grafista?
Recomendamos darse una vuelta por este canal para aprender un poquito acerca de cuáles son las herramientas necesarias, su utilidad y su relieve dentro del desarrollo.
Existen otros canales como #3DSMAX, en el que no os vamos a explicar cuál es el tema porque es bastante "gráfico", valga el juego de palabras... Otro canal que puede resultar interesante es #DIBUJO, que será dentro de una comparación, el equivalente al de #INFORMATICOS, en programación. Un canal de un ámbito muy general en el que tienen cabida muchos y muchos aficionados al tema, sea cual sea su conocimiento.
Y, por último, destacamos un canal de reciente aparición #GRAFISTAS, de los creadores de #PROGRAMACION_STRATOS, donde encontrarás a los grafistas de todos los grupos de programación de Stratos hablando de sus cosas. Y de vez en cuando hasta es posible que te topes con personajes de peso dentro de la industria del videojuego.

MAS DE LO QUE PODEIS IMAGINAR

Es cierto para aquellos que tenéis acceso a Internet y aún no conocéis el mundo de los canales en IRC, no dudéis en visitarlo sin falta pues seguro que no quedáis defraudados. Si tenemos que definir su contenido de alguna manera tendríamos que decir que encontrarás de todo para todos, aficionados, profesionales, todos tienen cabida en el canal.

Los que tenéis acceso a Internet y aún no conocéis el mundo de los canales en IRC, no dudéis en visitarlo

Y día a día el número de canales de este tipo aumenta, y aumenta, y aumenta (y nosotros tan contentos).



Consolas, of course!

No solo de PC's vive el hombre, ni mucho menos. En IRC también tienen cabida las consolas. Comenzando nuestro recorrido, entre la lista de canales podemos encontrar el canal #PLAYSTATION, especialmente dedicado para disfrutar de la charla entre los apasionados jugones de la increible consola de Sony. Y para los más curiosos, es posible conocer a otros usuarios que hayan trabajado con un kit de desarrollo de Net Yaroze, por ejemplo.
Aquí tenéis una pequeña muestra de la cantidad de canales dedicados a las consolas que podéis encontrar en IRC :

#N64-AREA	**SOLO PARA USUARIOS DE NINTENDO 64**
#N64PSX	**PARA FANS DE SONY Y DE NINTENDO**
#PLAYERS	**DE TODO UN POCO CONSOLA, EMULADORES, ...**
#PLAYSTATION	**SOLO PARA USUARIOS DE PSX**

Autor: Víctor Segura

Composición

Este mes describiremos las diferentes técnicas de tratamiento de una melodía y su posterior armonización. Técnicas indispensables para realizar composiciones que van más allá de lo que, en un principio, se puede imaginar.

Por lo general, cuando un músico compone una canción, lo primero que escribe sobre el pentagrama es la melodía. Una vez escrita toda la melodía estudia el modo en el que está y coloca, sobre los diferentes compases, los acordes de dicha melodía. A continuación, pasa a armonizar con los diversos instrumentos seleccionados para la canción.

Existen multitud de maneras de armonizar una melodía; la más común es la de seleccionar los acordes que la acompañarán

Este desarrollo puede llevar tanto tiempo como queramos, dependiendo de la dificultad de la canción y de los conocimientos teóricos del compositor. Claro está que cualquier persona que tenga buen oído puede hacer todo este trabajo, pero una base teórico-musical sólida será de gran utilidad, ya que lo que para un músico inexperto es un nuevo descubrimiento, para un músico experto y estudioso de la armonía (generalmente, armonía moderna), no es más que algo conocido y de fácil aplicabilidad.
Como decíamos al principio, el nacimiento de una canción es la melodía.

MELODIA PRINCIPAL

La melodía puede ser la base del éxito de una canción. Puede determinar la "fuerza de la canción", el sentido y muchas más sensaciones que se verán reforzadas con su posterior armonización. Es importante tener en cuenta que una misma melodía puede dar lugar a diferentes "momentos" dentro de una canción, situaciones que dependerán de la habilidad del compositor y, en otros casos, de la espectacularidad de los sonidos utilizados para interpretar la melodía.
Lo primero que tenemos que hacer es escribir una melodía. Un buen punto de partida puede ser una idea que se nos ha ocurrido en un determinado momento y, a partir de esa idea, dependiendo del rumbo que queramos que adquiera la canción, poder ir desarrollando melodías secundarias o bien seguir desarrollando esa primera melodía.

ACOMPAÑAMIENTO

Una vez terminada la melodía principal, procedemos a armonizarla. Existen multitud de maneras de armonizar una melodía; la más común es la de seleccionar los acordes que la acompañarán. Estos acordes pueden ponerse de varias maneras: un acorde por compás, más de uno por compás, un acorde por cada nota o una combinación de estas tres maneras. Para composiciones tipo rock suelen combinarse la primera con la segunda, mientras que para canciones al más puro estilo jazz, se combinan las tres incidiendo más en la tercera (un acorde por cada nota). La elección de los acordes será lo que nos indicará la tonalidad en la que estamos desarrollando la canción. Podemos cambiar de tonalidad aunque no cambiemos de melodía, si bien para realizar estos cambios tenemos que restringir las notas de la melodía dependiendo de la tonalidad en la que estemos, o bien, modificar las alteraciones de las notas de la melodía para acoplarlas con más facilidad a los cambios tonales.
Cuando hayamos decidido qué acordes acompañarán a nuestra melodía, procedemos a distribuir las notas sobre cada instrumento de la plantilla instrumental que tengamos a nuestra disposición.
Si, por ejemplo, decidimos que en el primer compás de nuestra canción vamos a poner una acorde Cmaj7, y hemos elegido una canción que va a tener un estilo rock, para la guitarra de acompañamiento pondremos la primera y la quinta del acorde (DO, SOL), para la guitarra solista pondremos la nota DO (de la melodía), para el teclado pondremos la tercera (MI) y la séptima (SI), mientras que para el bajo pondremos la primera (DO).

Existen diferentes formas de distribuir las notas del acompañamiento sobre los diferentes instrumentos. Además, dependerá del estilo que adoptemos. Para música orquestada tenemos lo que se conoce por concatenación o superposición, técnicas de armonización que vienen desarrolladas (tanto teórica como prácticamente) en el libro *Orquestación* de Walter Piston.

REESCRITURA

Cuando hayamos armonizado la canción, puede ser que tengamos que reescribir la melodía principal para adaptarla al acompañamiento. Para ello, tendremos que quitar notas, poner otras nuevas que son las que en determinados momentos de la canción dan sensación de inquietud, cambiar determinadas notas de lugar, etc... Existen muchas técnicas de reescritura y, entre ellas, está lo que se denomina variaciones melódicas.

VARIACIONES MELODICAS

Dependiendo de la base armónica y rítmica que hemos escogido para la melodía, podemos variar ésta con el fin de adaptarla a dicha base. Para ello, podemos utilizar las variaciones melódicas de la siguiente manera: cambiamos determinadas notas por otras siguiendo reglas que ponemos de antemano. Las reglas pueden ser las siguientes: decidimos qué notas son las que vamos a cambiar y por qué notas las vamos a sustituir. Por ejemplo, cada vez que haya un FA vamos a poner un DO y cada vez que haya un SOL vamos a poner un RE, es decir, sustituimos quintos grados por segundos grados. También podemos decidir que cada vez que aparezca un MI vamos a poner su tercera, (SOL), etc...

Existen diferentes formas de distribuir las notas del acompañamiento sobre los diferentes instrumentos

Asimismo, podemos modificar las melodías ascendentes por melodías descendentes, es decir, cada vez que aparezca una sucesión de notas que ascienden tonalmente por la escala que estemos utilizando reescribimos esa parte de la melodía manteniendo el ritmo, pero cambiando las notas de tal manera que siga una línea descendente. Cuando realizamos este

Notas de color para los bajos

Existen determinadas notas que dan un *color* especial a la canción. En alguna ocasión os habréis dado cuenta que el bajo no toca ni la nota principal del acorde (la primera), ni ninguna nota perteneciente al acorde. Probablemente utilice lo que se denominan "notas de color", que son notas que dotan de cierto colorido a la canción.

Imaginemos que, en el ejemplo que comentamos antes, cuando el bajo tocaba la primera del Cmaj7 cambiamos esa nota (DO) por su cuarta, es decir FA; probablemente, la sensación que experimentemos al oír el conjunto de todos los instrumentos va a ser distinta, de hecho puede que no suene tan bien como esperamos, pero eso es debido a que el resto de instrumentos (en concreto, la guitarra rítmica) no se ha modificado después de este cambio. En ocasiones, la propia guitarra toca también las notas de color, es decir, sustituiremos el **SOL** de la guitarra por el **FA**.

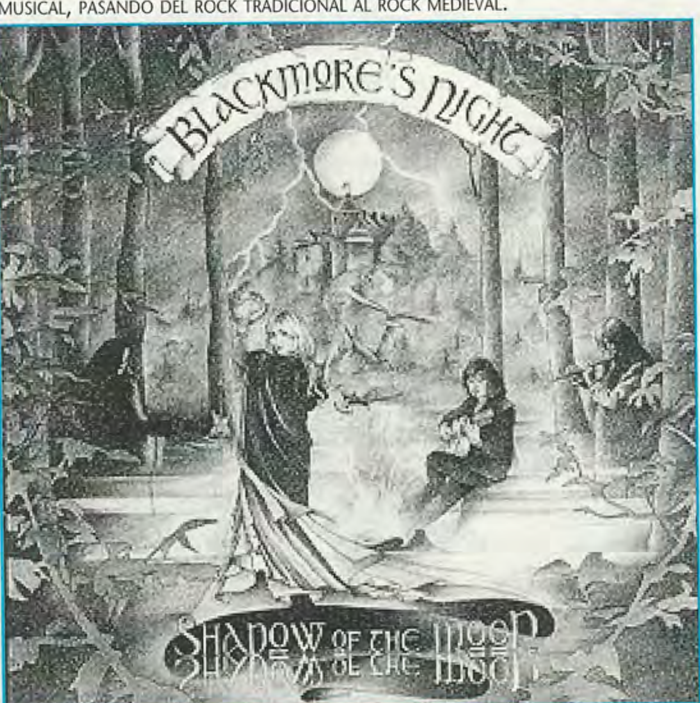
REGLA:

1.-Para los acordes mayores con séptima las notas de color se hará sobre la 4ª y 6ª del acorde, es decir, si tenemos un Cmaj7, las notas de color son FA (4ª) y LA (6ª).

2.- Para los acordes menores séptima las notas de color se harán sobre la 6ª (LA) y la 9ª(RE).

cambio podemos utilizar sustituciones como las descritas anteriormente; así, si tenemos una sucesión ascendente que es FA, SOL, RE, podemos reescribir DO, SOL, FA (tercera de RE), pero de forma descendente. Conviene practicar mucho esta técnica para familiarizarnos con ella y poder utilizarla de forma innata, no sólo para reescribir una melodía, sino para utilizarla directamente a la hora de escribir la melodía principal. Generalmente, los compositores recurren a hacerse fórmulas de variaciones para aplicarlas instantáneamente. Estas fórmulas pueden ser combinaciones de sustituciones y cambios ascendentes por descendentes o, como ocurre en el mundo musical, cualquier cosa que se os ocurra combinada con las sustituciones.

RITCHIE BLACKMORE EX-GUITARRA DE DEEP PURPLE, HA CAMBIADO SU ESTILO MUSICAL, PASANDO DEL ROCK TRADICIONAL AL ROCK MEDIEVAL.



VARIACIONES RITMICAS

Además de las variaciones melódicas, existen las variaciones rítmicas que consisten en interpretar la misma melodía cambiando el tiempo y el ritmo de las diferentes notas que la componen. Podemos sustituir una corchea por una semicorchea, añadir un silencio entre dos notas consecutivas, trasladar la melodía un tiempo, utilizar tresillos usando aproximaciones cromáticas entre notas (puede ser una técnica de reescritura melódica), en vez de utilizar corcheas, etc...

¿UN CICLO INFINITO?

Puede ser que, en nuestro afán de superarnos, escribamos la melodía, armonicemos, variemos la melodía, rearmonicemos, reescribamos la melodía, volvamos a armonizar.... Conviene ser críticos con nuestro trabajo, pero también saber hasta dónde llegar con dicha crítica. Tened en cuenta que las variaciones y rearmonizaciones se utilizan en determinados puntos de la canción y su finalidad es la de mejorar la canción en esos puntos. Si nos proponemos llegar más allá de nuestras posibilidades, lo único que conseguiremos es embarcarnos en un viaje sin retorno y no terminar jamás nuestra canción, ya que cada vez que terminamos con la reescritura de la melodía volvemos a armonizar y, así, sucesivamente.

DIFERENTES ESTILOS SOBRE UNA MISMA MELODIA

Sobre una melodía podemos realizar diferentes estilos musicales aplicando variaciones melódicas, variaciones rítmicas y armonizaciones diferentes. De hecho, en la música rock no se suelen utilizar las notas de color en los bajos, mientras que en jazz (en las corrientes más modernas), es una de las características más comunes. En la música orquestada, sin embargo, se utilizan variaciones melódicas.

Como hemos dicho en diferentes números de esta revista, el estilo de una canción está definido, principalmente, por la plantilla instrumental que seleccionamos

Como hemos dicho en diferentes números de esta revista, el estilo de una canción está definido, principalmente, por la plantilla instrumental que seleccionemos. Pero una vez seleccionada la plantilla, la manera de armonizar la melodía será clave para personalizar el estilo elegido, incluso podemos hacer sobre una misma base una canción con diferentes cambios de estilo, caracterizados por cambios armónicos, rítmicos y variaciones (incluso arreglos) sobre la melodía principal.

EL PROXIMO MES

En el artículo siguiente estudiaremos las pentatónicas mayores y menores. Con su ayuda podremos desarrollar solos (concretamente de guitarra), al más puro estilo ACDC para componer temas de rock que no tengan nada que envidiar a los solos de los guitarristas más virtuosos del momento.

Desarrollo de una melodía

Podemos combinar la melodía principal y sus variaciones de muchas maneras, dependiendo del estilo que estemos utilizando y de la intención del compositor. Un ejemplo de como podemos hacer una canción a partir de una melodía y de sus variaciones puede seguir el esquema siguiente:

Melodía original, variaciones sobre la melodía, melodía principal y melodía y variaciones juntas.

